Int. J. on Recent Trends in Engineering and Technology, Vol. 8, No. 2, Jan 2013

Design and Characterization of SDRAM Controller IP Core with Built In ECC Module

Arathy S[1], Nandakumar.R[2], and Hima Sara Jacob[2]
[1]NIELIT, VLSI Design Group, Calicut, India
Email: arathysnair89@gmail.com
[2]NIELIT, VLSI Design Group, Calicut, India
Email: {nanda24x7, hima.sara24}@gmail.com

***Abstract*—In modern digital systems large capacity and data transfer rate is required. Synchronous DRAM (SDRAM) became the memory of choice due to its speed, burst access and pipeline features. A Controller is required to provide proper commands for SDRAM initialization, read/write accesses and memory refresh. In semiconductor memories there are chances of errors. To ensure reliable data storage, an error correction and detection scheme is required. This paper describes the architecture design and characterization of SDRAM Controller IP core with built in Error Correcting Codes (ECC) module which is vendor neutral. The design is described using Verilog HDL, simulated using ModelSim and prototyped in Altera® platform FPGA. Resource utilization and power analysis was done using Altera® Quartus II. Hardware test results are obtained from Signal Tap Logic Analyzer.**

***Index Terms*—Synchronous DRAM, Error Correcting Codes, IP core, Power analysis, Resource utilization.**

I. Introduction

Synchronous DRAMs (SDRAMs) become the memory of choice in many digital systems because it provides a significant improvement in bandwidth performance over traditional asynchronous DRAMs such as "FPM" (Fast Page Mode) and "EDO" (Extended Data Out). In Synchronous DRAMs input address, data, and control signals are typically latched on the positive edge of the clock signal. SDRAMs offer several features such as multiple internal banks, burst mode access, and pipelining of operation executions, that helps to improve bandwidth performance.

There are two popular types of SDRAM in market. The most common single data rate (SDR) SDRAM transfers data typically on the rising edge of the clock. The other is the double data rate (DDR) SDRAM [5] which transfers data on both the rising and falling edge to double the data transfer throughput. Other than the data transfer phase, the different power-on initialization and mode register definitions; these two SDRAMs share the same command set and basic design concepts. This paper describes a design that is targeted for SDR SDRAM. However, due to the similarity of SDR and DDR SDRAM, this design can also be adapted for a DDR SDRAM controller. For benchmarking purpose, the Micron® SDR SDRAM MT48LC8M16A2 is chosen as a target for this design. In semiconductor memories there are chances of errors. Therefore an error correction and detection scheme can be used to provide reliable data storage. Also, this design has been verified by using memory simulation model.

The section II of this paper is a tutorial review of Synchronous DRAMs. The section III describes the design of the proposed SDRAM Controller with ECC module. The section IV describes the simulation results. The section V describes the implementation results of the proposed controller.

II. Synchronous Dram Review

SDRAM, or Synchronous Dynamic Random Access Memory is a form of semiconductor memory that can run at faster speeds than conventional DRAM. Since SDRAM has a synchronous interface, it has an internal finite state machine that pipelines incoming instructions. Thus the speed of operation is much higher.

The Micron® 128Mb [4] SDRAM referenced in this paper is a high-speed CMOS, dynamic random access memory containing 134,217,728 bits. It is internally configured as a quad-bank DRAM with a synchronous interface. Each of the 33,554,432-bit banks is organized as 4,096 rows by 512 columns by 16 bits. The 128Mb SDRAM is designed to operate in 3.3V memory systems. All inputs and outputs are LVTTL-compatible.

III. Proposed Controller Design

A. Introduction

The proposed SDRAM Controller is designed to work with a standard memory from Micron Technology® with series MT48LC8M16A2™. It has a user interface end on one side and 128Mb SDRAM on the other end. The design is coded in Verilog HDL. The controller offers facility for programmable burst lengths of 1, 2, 4, and 8, programmable CAS latency of 2 and 3.

Initialization should be done before applying any normal operation. Read and Write should be performed only after the initialization. The proposed controller design automatically performs all the initialization procedures [1, 2, 4]. ECC block consists of encoder and decoder-corrector, which can detect and correct single-bit errors and detect double-bit errors.

B. Block Diagram

The block diagram of proposed SDRAM Controller is shown in figure 1. The main function of the controller is to

DOI: 01.IJRTET.8.2.29

convert user commands to commands that can be understandable by the memory [6]. Another is to detect and correct single-bit errors and detect double-bit errors[3]. The main blocks include datapath, finite state machine, initialization control block and ECC block.

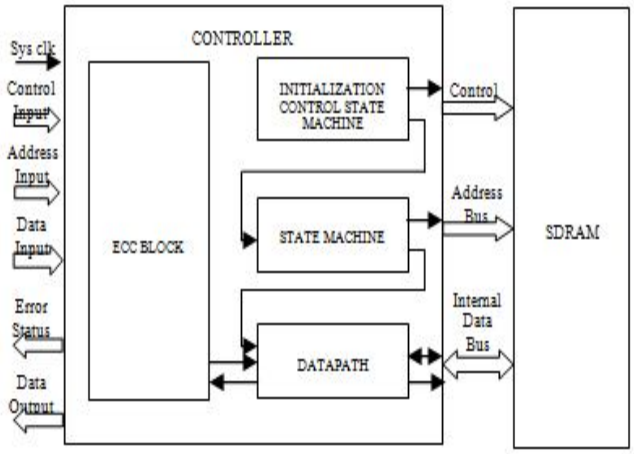

Figure 1. Controller block diagram

The inputs like command (cmd), reset are collectively shown as control input. The memory address is given through the address input. The data during write operation is given through the data input. Data during read operation is obtained through data output. Error status is also obtained as an output.

C. Pin Description

The input output diagram of controller is shown in figure 2.

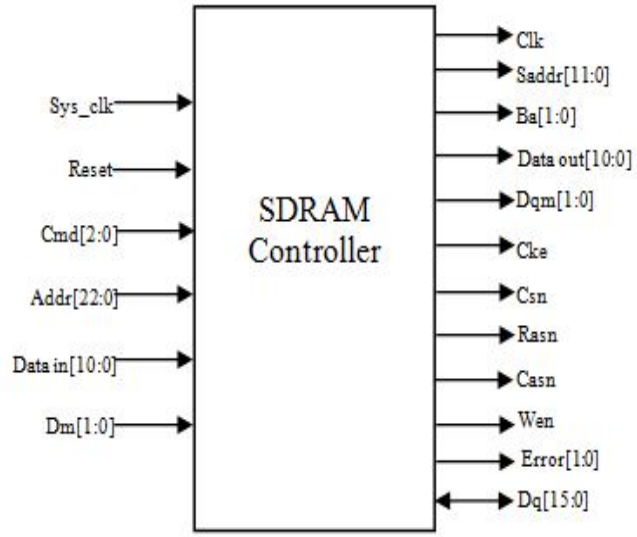

Figure 2. Pin out diagram of controller

Type and the function of all the pins are described in table I.

D. Detailed Architecture

The detailed architecture of the proposed controller is shown in fig 3.

E. Initialization Control State Machine

According to architecture and working principle of SDRAM [4], the controller design uses two Finite State Machines to implement timing-logic control [1,2]. The SDRAM must be powered up and initialized in a predefined manner before any normal operation. Automatic initialization

TABLE I. PIN DESCRIPTION

PIN	TYPE	DESCRIPTION
Sys Clk	Input	Input clock to the controller. Used to generate the clock (clk) to the memory.
Reset	Input	System reset - Used to reset the total system.
Cmd [2:0];	Input	Command input which distinguishes between different operations like Precharge, refresh, load mode register, active, read and write.
Data in [10:0]	Input	11-bit data input to the controller which is used as data during write operation.
Data out [10:0]	Output	11-bit data output from the controller during read operation.
Addr [23:0]	Input	23-bit Address bus. From this 23-bit address row address, column address and bank address is decoded.
Dm [1:0]	Input	Dm is used to mask input data during write cycle and output data during read cycle.
Clk	Output	CLK is driven by the system clock. Typical values are 100MHz and 133MHz.
Saddr [11:0]		Address input A0–A11 are sampled during the ACTIVE command (row address A0–A11) and READ/WRITE command (column-address A0–A8).
Ba [1:0]	Output	Bank address is used to select the bank
Cke	Output	CKE activates (HIGH) and deactivates (LOW) the CLK signal.
Csn	Output	CS enables (LOW) and disables (HIGH) the command decoder. All commands are masked when CS is registered HIGH.
Rasn	Output	Command inputs WE, CAS, and RAS (along with CS) define the command being entered.
Casn	Output	
Wen	Output	
Error [1:0]	Output	Error is an output signal used to display the error status. If it is "00", no error has occurred. If it is "10", single-bit error has occurred and corrected. If it is "01", double-bit error has occurred and not corrected. Value "11" is not possible
Dqm [1:0]	Output	DQM is an output mask signal for write accesses and an output enable signal for read accesses. Input data to the memory is masked when DQM is sampled HIGH during a WRITE cycle. The output buffers are placed in a High-Z state (2-clock latency) when DQM is sampled HIGH during a READ cycle. DQM is generated from Dm signal.
Dq [15:0]	Inout	Bidirectional data bus which is connected to the bidirectional data bus of memory. During write operation Dq is directed from controller to the memory. During read operation Dq is directed from memory to the controller

DOI: 01.IJRTET.8.2.29

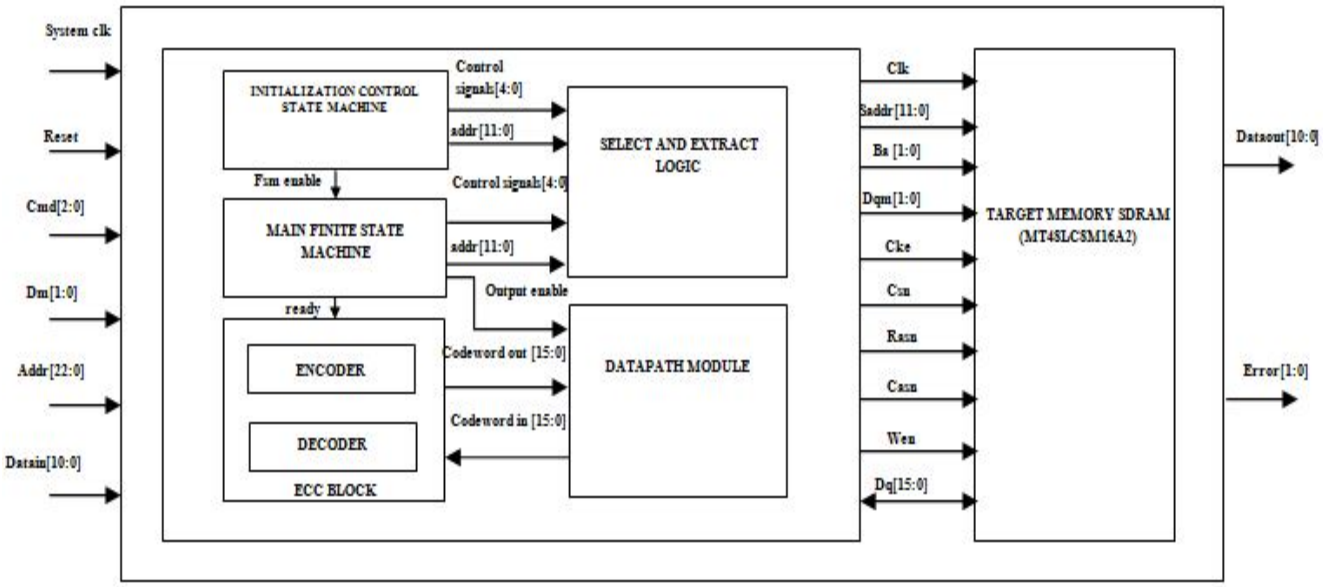

Figure 3. Architecture of the controller

is carried out by this state machine. The main processes in initialization is shown in figure 4.

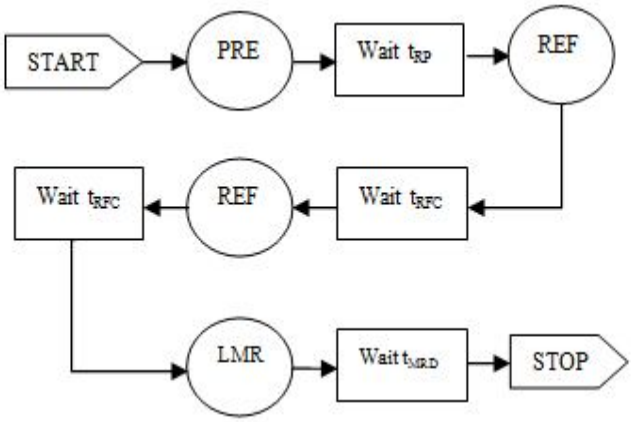

Figure 4. Initialization process flow

Initialization process consist of sequence of operations. First Precharge all banks(PRE) should be done. Precharging is required to deactivate all banks and put them in idle state. After precharging wait for t_{RP} (150 ns) period and execute Auto Refresh(REF). After autorefresh wait for t_{RFC} (495 ns) period and again apply Auto Refresh. Again wait for t_{RFC} period and then Load Mode Register(LMR) command should be issued. With this command some special attributes are set (i.e. burst packet length , access type, CAS latency and other). Then wait for t_{MRD} (15 ns) period. After the initialization, the SDRAM goes into the idle state, which is the initial state of the main FSM which is explained below.

F. Main Finite State Machine

Another state machine is the main FSM or the core of the controller which is shown in figure 5. The FSM generates control signals corresponding to the state. Transition between different states is according to the command input.

The controller awakens in the IDLE state and then changes to Precharge All, Precharge Selected, Load Mode Register, AutoRefresh, or Active depending on the system

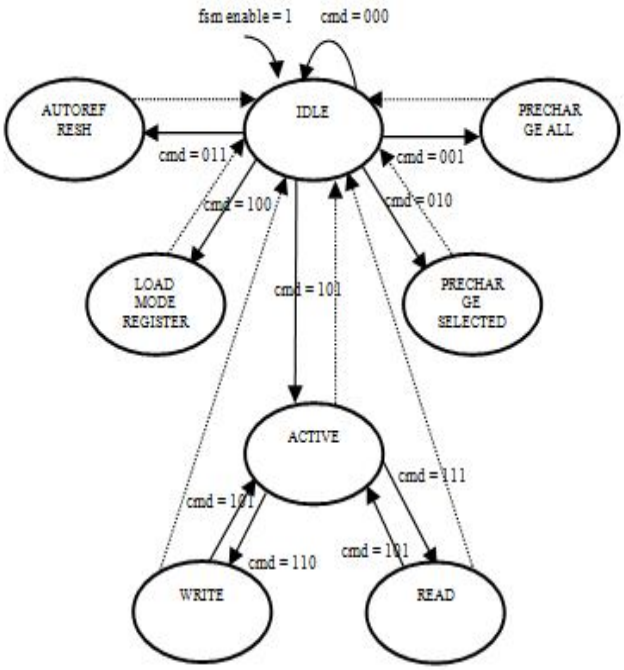

Figure 5. Main FSM

command. The dashed lines in the state machine diagram indicate the automatic transfer. For Read and Write, the controller first goes into Active state. During active state row address and bank address is decoded from the address input. Controller enters write state when command is 110 and read state when command is 111.

G.. Datapath Circuit

The data flow design between the SDRAM and the system interface is shown in figure 6. The datapath circuit consists of four number of 16-bit D flip flops and a tristate buffer. Flip flop is used to shift the data. The data passing through the datapath circuit is either the codeword from the ECC Encoder or codeword from the memory. Tristate buffer is used to determine the direction of bidirectional data bus. Tri state

DOI: 01.IJRTET.8.2.29

buffer has a control input "oe" which determines whether it is a read or write operation. It is obtained as an output from the main finite state machine.

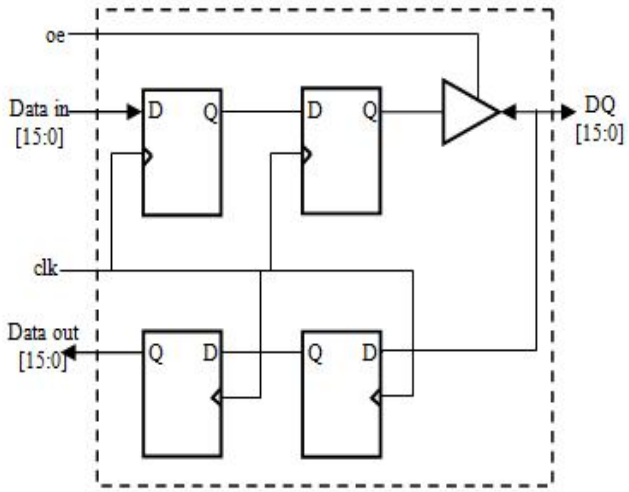

Figure 6. Datapath circuit of the controller

H. *Error Correcting Codes (ECC) block*

Hamming Codes can be used to detect and correct single-bit errors and detect double-bit errors. It is relatively simple yet powerful ECC code. It involves transmitting data with multiple check bits(parity) and decoding the associated check bits when receiving data to detect errors. ECC block comprises an encoder and a decoder-corrector which can detect and correct single-bit errors and detect double-bit errors.

The number of parity bits, *m*, needed to detect and correct a single bit error in a data string of length *n* is given by the following equation(1) [3].

$$m = log_2 n + 1 \quad (1)$$

The ECC block uses the Hamming code with an additional parity bit, which can detect single and double-bit errors, and correct single-bit errors[3,7]. The extra parity bit applies to all bits after the Hamming code check bits have been added. This extra parity bit represents the parity of the codeword. If one error occurs, the parity changes, if two errors occur, the parity stays the same. In general the number of parity bits, *m*, needed to detect a double-bit error or detect and correct a single-bit error in a data string of length *n*, is given by the following equation(2) [3].

$$m = log_2 n + 2 \quad (2)$$

The target memory is 16 bit wide. Therefore sum of data bits and parity bits should not exceed 16 bits. 16-bit codeword can provide error correction and detection to 11-bit data. For an 11-bit message there are 11 possible single-bit errors. Therefore hamming code (16,11) can be used in this case. (16,11) hamming code means 11 bit data, 5 parity bits and 16 bit codeword. Hamming codeword is a concatenation of the original data and the parity bits.

ECC block comprises ECC encoder and ECC decoder-corrector. Encoder converts the 11-bit input message into 16-bit codeword by placing parity bits in power-of-two positions. This 16-bit codeword is stored in the memory. Decoder-corrector detects any error in the received codeword by calculating syndrome bits and corrects single-bit errors by using a mask.

ECC Encoder

The encoder takes the input data and encodes the message into a (11 + 5) bit codeword. It uses the principle of Hamming Code with an additional parity bit. The parity bits can be calculated according to the equations given in table II. The parity bit P[0] is generated by XORing the data bits that have "1" at position "k" in the address field xxxk [3,8]. Similarly P[1], P[2] and P[3] are also calculated. P[4] is generated by XORing all the data bits and previously calculated parity bits.

TABLE II. PARITY BITS CALCULATION

Parity-P[4:0]	Equation [Input Data – d[10:0]]
P[0]	d[0] ^ d[1] ^ d[3] ^ d[4] ^ d[6] ^ d[8] ^ d[10]
P[1]	d[0] ^ d[2] ^ d[3] ^ d[5] ^ d[6] ^ d[9] ^ d[10]
P[2]	d[1] ^ d[2] ^ d[3] ^ d[7] ^ d[8] ^ d[9] ^ d[10]
P[3]	d[4] ^ d[5] ^ d[6] ^ d[7] ^ d[8] ^ d[9] ^ d[10]
P[4]	d[0] ^ d[1] ^ d[2] ^ d[3] ^ d[4] ^ d[5] ^ d[6] ^ d[7] ^ d[8] ^ d[9] ^ d[10] ^ P[3] ^ P[2] ^ P[1] ^ P[0]

The parity bits are placed in power-of-two positions (1,2,4,8). The position of parity bits and information bits are shown in figure 7.

P[4]	P[0]	P[1]	d[0]	P[2]	d[1]	d[2]	d[3]
0	1	2	3	4	5	6	7
P[3]	d[4]	d[5]	d[6]	d[7]	d[8]	d[9]	d[10]
8	9	10	11	12	13	14	15

Figure 7. Codeword (16-bit)

ECC Decoder-Corrector

The decoder-corrector is a self-contained entity for messages of 11 bit. The decoder creates a syndrome and sends it with the received message to the corrector. The corrector extracts the syndrome bits and analyse it. By analysing the value of syndrome bits, the corrector detects single-bit and double-bit errors and corrects the single bit errors of the received message according to the syndrome. The process flow diagram of decoder-corrector block[7] is shown in figure 8.

In the syndrome computation[3] step, decoder computes five syndromes using the 16-bit codeword received from the memory. The syndromes S[0] to S[4] are computed by XORing the encoder parity bits P[0] to P[4] and decoder parity check bits C[0] to C[4]. Encoder parity bits are obtained from the power-of-two positions in the codeword received. Decoder parity check bits are obtained by calculating the parity of the bits in the received codeword. Equation for syndrome is given below.

DOI: 01.IJRTET.8.2.29

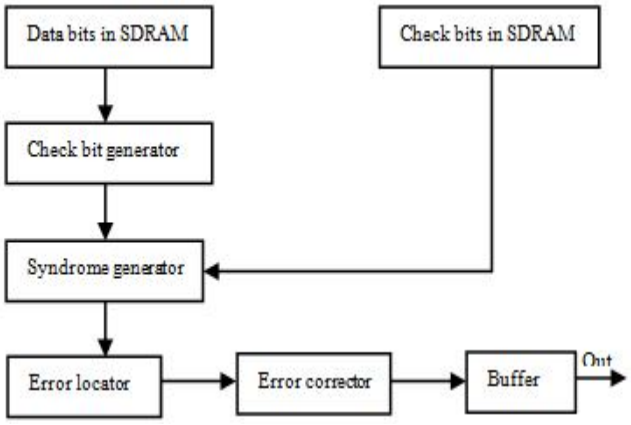

Figure 8. Decoder process flow

$$Syndrome <= check \wedge parity \quad (3)$$

The corrector module is responsible for error diagnosis and error correction. Principle of error correction and error diagnosis is explained below.

Error Diagnosis

The ECC block can diagnose if a one- or two-bit error has occurred on the data from memory. To indicate error status a 2-bit error signal is provided. Error[1] indicate single bit error and error[0] indicate double bit error. Diagnosis table is given in table III.

TABLE III. DIAGNOSIS TABLE

Error[1]	Error[0]	Diagnosis
0	0	There is no error on the message on the output.
1	0	There was one error on the codeword the message is equivalent to the original.
0	1	There are two errors on the codeword no correction have been made.
1	1	Not possible.

Error diagnosis is done by examining the syndrome bits. If the S[4] of received syndrome is '0', the corrector checks whether the bits S[3:0] is '0000' or not. If it is '0000', it is inferred that no error has occurred. If bits S[3:0] is not '0000', it is inferred that a double bit error has occurred. If the S[4] is '1', then it is inferred that a single bit error has been occurred.

Mask Generation and Error Correction

Mask is used to correct the message. 11-bit mask is generated from the value of syndrome. The value of syndrome indicates the position of erroneous bit. The corresponding bit position of the mask is set high. The corrected message is calculated by XORing the mask and the decoded data from the decoder.

$$Corrected\ output\ data <= mask \wedge decoded\ data \quad (4)$$

IV. SIMULATION RESULT

The Controller along with memory can be simulated using Modelsim software. The functional simulation result of the wrapper module is shown in Fig. 9. Two clock signals are shown sys_clk and clk. Former is the clock input to the controller. Latter is the clock input to the memory. It is generated by the controller. The designed IP core can be further prototyped using Altera DE2 FPGA board[9].

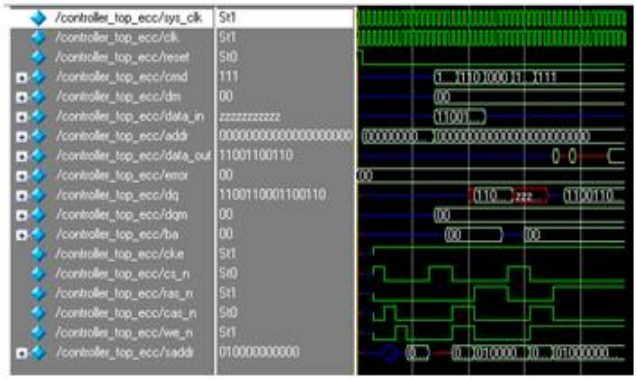

Figure 9. Simulation result

Note on simulation result :

Reset : (sys_clk = 1'b1, clk = 1'b1, reset = 1'b1, cmd = 3'bxxx, dm = 2'bxx, data_in = 11'bxxxxxxxxxxx, addr = 23'bxxxxxxxxxxxxxxxxxxxxxxx, data_out = 11'bzzzzzzzzzzz, error = 2'b00)

Initialization : (sys_clk = 1'b1, clk = 1'b1, reset = 1'b0, cmd = 3'bxxx, dm = 2'bxx, data_in = 11'bxxxxxxxxxxx, addr=23'b00000000000000000010001, data_out = 11'b00000000000, error = 2'b00)

Activate : (sys_clk = 1'b1, clk = 1'b1, reset = 1'b0, cmd = 3'b101, dm = 2'b00, data_in = 11'b11001100110, addr = 23'b00000000000000000000000, data_out = 11'b00000000000, error = 2'b00)

Write : (sys_clk = 1'b1, clk = 1'b1, reset = 1'b0, cmd = 3'b110, dm = 2'b00, data_in = 11'b11001100110, addr = 23'b00000000000000000000000, data_out = 11'b00000000000, error = 2'b00)

Idle : (sys_clk = 1'b1, clk = 1'b1, reset = 1'b0, cmd = 3'b000, dm = 2'b00, data_in = 11'bxxxxxxxxxxx, addr = 23'bxxxxxxxxxxxxxxxxxxxxxxx, data_out = 11'b00000000000, error = 2'b00)

Activate : (sys_clk = 1'b1, clk = 1'b1, reset = 1'b0, cmd = 3'b101, dm = 2'b00, data_in = 11'bxxxxxxxxxxx, addr = 23'b00000000000000000000000, data_out = 11'b00000000000, error = 2'b00)

Read : (sys_clk = 1'b1, clk = 1'b1, reset = 1'b0, cmd = 3'b111, dm = 2'b00, data_in = 11'bxxxxxxxxxxx, addr = 23'b00000000000000000000000, data_out = 11'b11001100110, error = 2'b00)

V. IMPLEMENTATION RESULTS

The controller module is tested by using Altera® DE2 FPGA board. Sys_clk is taken from the board itself. Reset is given as trigger input. Altera Quartus® II is used to synthesis the design. Signal Tap® II Logic Analyzer is used to take hardware test result from FPGA. Resource utilization and Power analysis result is listed in table IV and table V respectively.

DOI: 01.IJRTET.8.2.29

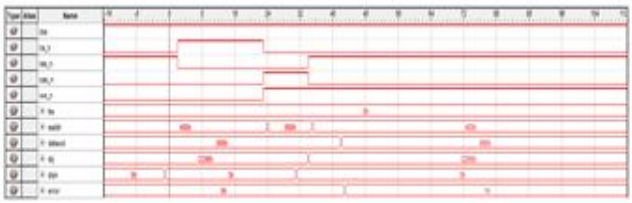

Figure 10. Hardware test result

TABLE IV. RESOURCE UTILIZATION TABLE

RESOURCE	USAGE
Estimated Total logic elements	1,376
Total combinational functions	696
Logic element usage by number of LUT inputs	
-- 4 input functions	367
-- 3 input functions	173
-- 2 input functions	156
Logic elements by mode	
-- normal mode	656
-- arithmetic mode	40
Total registers	1116
-- Dedicated logic registers	1116
-- I/O registers	0
I/O pins	52
Total memory bits	6400
Maximum fan-out node	sys_clk
Maximum fan-out	781
Total fan-out	6403
Average fan-out	3.34

Resource Utilization table summarizes usage statistics for resources including logic elements, registers, I/O pins, memory blocks, interconnect usage, and fan-out.

TABLE V. POWER ANALYSIS TABLE

PARAMETER	RESULT
Total Thermal Power Dissipation	115.87 mW
Core Dynamic Thermal Power Dissipation	0.00 mW
Core Static Thermal Power Dissipation	79.94 mW
I/O Thermal Power Dissipation	35.92 mW

Power Analysis table gives details about the core dynamic thermal power dissipation, core static thermal power dissipation, I/O thermal power dissipation and total thermal power dissipation.

CONCLUSION

This paper describes the the design, simulation and characterization of synthesizable SDRAM Controller IP core with built in ECC module. The design was developed using Verilog HDL. It can be easily modified for different system design requirements. The proposed design has been tested by implementing the design on Altera DE2 board which uses Cyclone-II device.

REFERENCES

[1] Tomasz Szymanski, Rafak Kiekbik, Andrzej Napieralski, "SDRAM controller for real time digital image processing systems." CAD Systems in Microelectronics, 2001. CADSM 2001. Proceedings of the 6th International Conference.

[2] Qiu Daqiang, Hu Bing, Li Dandan " Design of SDRAM Controller in High-Speed Data Acquisition Based on PCI Bus." The Eighth International Conference on Electronic Measurement and Instruments 2007.

[3] Altera, "DDR and DDR2 SDRAM ECC Reference Design". 2006.

[4] MICRON Technology Inc.,"Synchronous DRAM : MT48LC8M16A2 Data Sheet.",2012 available online at http://download.micron.com/pdf/datasheets/dram/sdram/128msdram.pdf

[5] MICRON Technology Inc., "DOUBLE DATA RATE (DDR) SDRAM: MT46V32M16 Data Sheet.", 2012 availble online at http://download.micron.com/pdf/datasheets/dram/ddr/512MBDDRx4x8x16.pdf

[6] Xilinx Inc., XAPP134 "Synthesizable High Performance SDRAM Controller." 2000.

[7] W. Gao and S. Simmons, "A study on the VLSI implementation of ECC for embedded DRAM," Electrical and Computer Engineering,2003. IEEE CCECE 2003. Canadian Conf., Vol. 1, pp. 203-206, May 2003.

[8] M. Y. Hsiao, "A class of optimal minimum odd-weight-column SEC_DED codes," IBM J.Res. Develop., vol. 14,pp. 395-401, July 1970.

[9] Altera DE2 Development Board User Manual available online at ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf

DOI: 01.IJRTET.8.2.29